Multi-Dimensional Bisection Method

Efficient computation of stability chatrs

Daniel Bachrathy, PhD

assistant professor
Department of Applied Mechanics, Budapest University of Technology and Economics

Multi-Dimensional Bisection Method (MDBM) is an efficient and robust root-finding algorithm, which can be used to determine whole high-dimensional submanifolds (points, curves, surfaces…) of the roots of implicit non-linear equation systems, even in cases, where the number of unknowns surpasses the number of equations.

$$f_i(x_j) = 0 \quad i = 1...k \quad j = 1...l, \quad k \leq l$$

This method is an alternative to the contour plot or to isosurfaces in higher dimension, however, it has as the advantage of being able to handle multiple functions at once.
In addition, it uses far less function evaluation than the brute-force approach, making it much faster and more memory efficient, especially for complex tasks.

Currently a Matlab package and a Julia package is available

- https://github.com/bachrathyd/MDBM-Matlab
- https://github.com/bachrathyd/MDBM.jl

Introduction

The bisection method - or the so-called interval halving method - is one of the simplest root-finding algorithms which is used to find zeros of continuous non-linear functions. This method is very robust and it always tends to the solution if the signs of the function values are different at the borders of the chosen initial interval.

Geometrically, root-finding algorithms of **f**(**x**)=0 find one intersection point of the graph of the function with the axis of the independent variable. In many applications, this 1-dimensional intersection problem must be extended to higher dimensions, e.g.: intersections of surfaces in a 3D space (volume), which can be described as a system on non-linear implicit equations. In higher dimensions, the existence of multiple solutions becomes very important, since the intersection of two surfaces can create multiple intersection curves.

MDBM algorithm can handle automatically:

- multiple solutions
- arbitrary number of parameter (typically: 3-6)
- arbitrary number implicit equations
- constraints in the parameter space
- handle degenerated functions
- first and second order interpolation (and convergence rate)
- provides the gradients of the equations for the roots
- refinement the selected region only

- interpolation for higher resolution plotting

Motivation: Compute the stability chart of a Dynamical System

During my studies and research, I have to determine stability charts of models described by delayed differential equations, which are typically formed as a "2 implicit equation / 3 parameter" problem. The two equation is related to the real and imaginary part of the characteristic equation. Two parameter determines the stability chart and the third parameter is the vibration frequency (of the trial function).

I have faced the difficulty that there is no applicable solution in any available software (e.g.: Mathematica, Matlab,...) which could easily be used in engineering problems. The continuation method in a bifurcation analysis could be a good solution, however, it is not automatic,requires many human interaction, fail to find closed curves. Due to this reason, I have started to develop the Multi-Dimensional Bisection Method in 2006 in Matlab, and I have been improving it since, by adding new features from time to time. The combination of the MDBM with the Semi-Discretization method, Multi-Frequency solution of other spectral methods can lead to an efficient stability chart calculation.

Further more, higher dimensional rooth-finding problems can be found in many different field of science, just to mention a few published application the the MDBM:

- stability chart computation
- stabilizability diagram
- robust stability calculation
- differential geometry (isolines, isosurfaces in higher dimensions)
- analysis of linkages (mechanical: workspace of robots)

Citing

The software in this ecosystem was developed as part of academic research. If you use the MDBM.jl package as part of your research, teaching, or other work, I would be grateful if you could cite my corresponding publication: https://pp.bme.hu/me/article/view/1236/640

Bachrathy Dániel, Stépán Gábor, Bisection method in higher dimensions and the efficiency number. PERIODICA POLYTECHNICA-MECHANICAL ENGINEERING 56:(2) pp. 81-86. (2012) DOI: 10.3311/pp.me.2012-2.01

Web:

https://www.mm.bme.hu/~bachrathy/research_EN.html

MDBM - Matlab Package

First of all, add the folder of the downloaded program files (*MDBM-Matlab-master/code_folder/*) to your path:

- Linux

```
addpath("/home/..../MDBM-Matlab-master/code_folder/")
```

- Windows

```
addpath("C:\....\MDBM-Matlab-master\code_folder\")
```

Basics:

- Create the parameter space with the corresponding initial mesh in a following form

```
ax=[];
ax(1).val=linspace(-3,3,8);% x parameter
ax(2).val=linspace(-3,3,8);% y parameter
ax(3).val=linspace(-3,3,8);% z parameter
```

- Create a *vectorized* function which will provide the implicit function values for the parameter space Create function in a separated `.m` file

```
function H=fval_parmeterdim3_codim2(ax)
% ax contains the coordinate in column vector (first index of ax) of
the evaluation points (second index of ax)
H=zeros(2,size(ax,2)); % initialization of the output
for k=1:size(ax,2) % iterate through all the point
    x=ax(1,k);% x coordinate of the k-th point
    y=ax(2,k);% y coordinate of the k-th point
    z=ax(3,k);% z coordinate of the k-th point
    H(1,k)=x^2+y^2+z^2-2^2; % first equation: implicit equation of a
sphere
    H(2,k)=sin(x)-y;% second equation: implicit form of y=sin(x)
end
```

or create an anonymous function

```
fun=@(ax)[...
ax(1,:).^2+ax(2,:).^2+ax(3,:).^2-2^2.5   ;...% first equation:
implicit equation of a sphere
sin(ax(1,:))-ax(2,:)...% second equation: implicit form of y=sin(x)
];
```

- Define the number of iteration

```
Niteration=5;
```

The initial grid will be devided *Niteration* times. So the number of points in the final grid it ~$2^{Niteration}$ times more than in the initial grid. Take care, the large values can easily lead to memory problem. Start

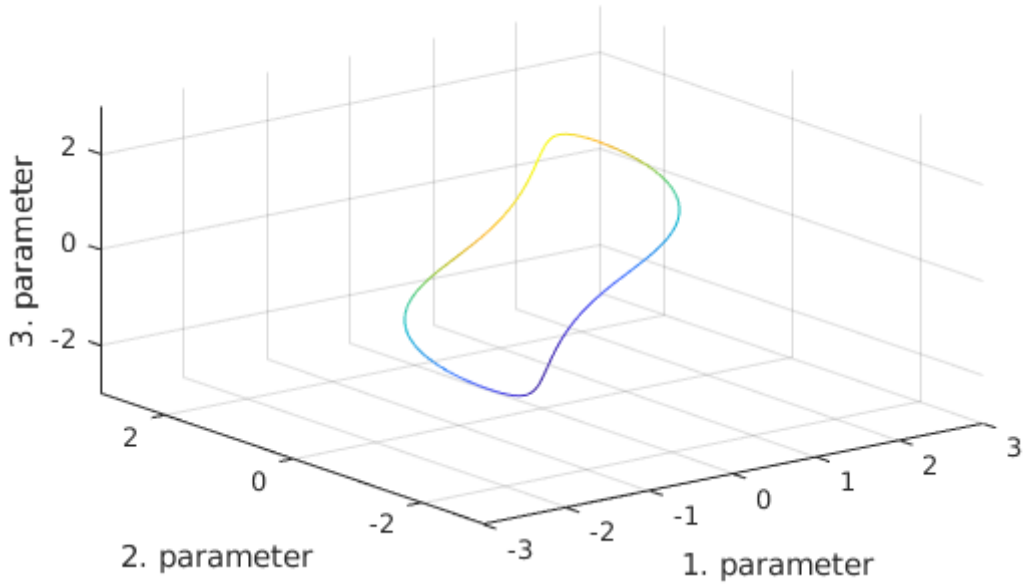
around 3.

- Run the Multi-Dimensional Bisection Method

```
mdbm_sol=mdbm(ax,fun,Niteration);
```

- The solution can be plotted direcly

```
plot_mdbm(mdbm_sol);
```



Examples for the delayed oscillator

A simple 3 parameter/ 2 equation problem of the delayed oscillator:

$x''(t)+\kappa x'(t) + \delta x(t) = b x (t-\tau)$
$x(t)=A e^{i \omega t}$
$D(\delta,b,\omega)=-\omega^2+i\kappa \omega+\delta-b e^{-i \omega \tau}$

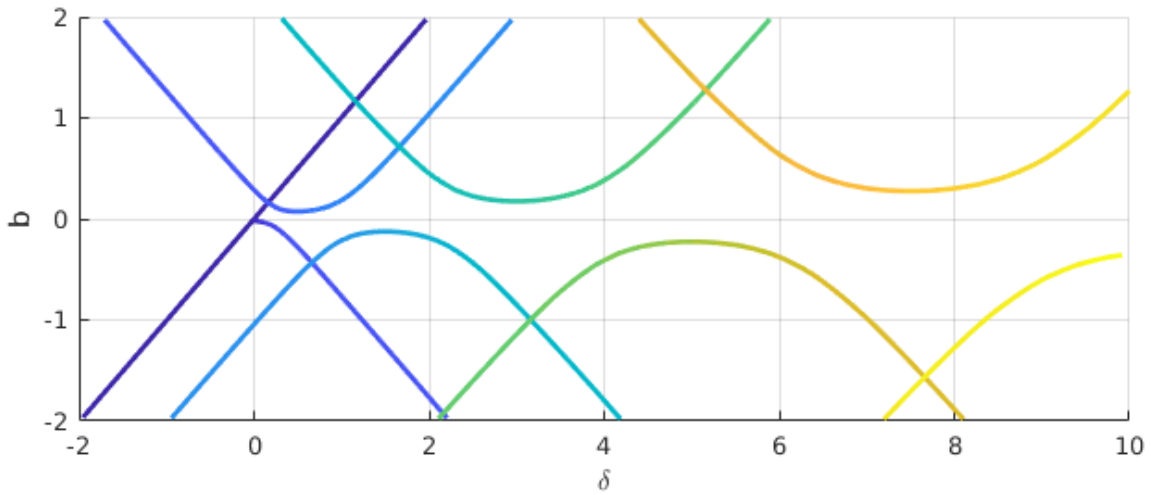
- The *vectorized* function for the characteristic equation, created in a separated `.m` file

```
function CharEQ_Re_Im=fval_complex_7_delay_oscill_cont(ax,par)
CharEQ_Re_Im=zeros(2,size(ax,2));%initialization of the output
for kax=1:size(ax,2)
    delta=ax(1,kax);%proportional
    b=ax(2,kax);%delayed
    om=ax(3,kax);%critical frequency

    tau=2*pi;
    % par.kappa is the constant damping
    D=(1i*om)^2+(par.kappa)*1i*om+delta-b*(exp(-1i*om*tau));
```

```
    CharEQ_Re_Im(1,kax)=real(D);
    CharEQ_Re_Im(2,kax)=imag(D);
end
end
```

- Code to compute the stability chart:

```
%% definition of the parameter space
ax=[];
ax(1).val=linspace(-2,10,5);%delta
ax(2).val=linspace(-2,2,5);%b
ax(3).val=linspace(-0.01,5,10);%critical frequency

%% extra constant parameters
par=[];
par.kappa=0.1;

par.interpolationorder=1;
mdbm_sol=mdbm(ax,'fval_complex_7_delay_oscill_cont',4,[],par);

figure(103),clf
ghandle=plot_mdbm(mdbm_sol);
set(ghandle, 'LineWidth',2)
view(2)
xlabel('\delta')
ylabel('b')
```



Examples in the Repository

For advanced options go through the extensive examples:

Basic examples

Use them as a template for your problem based on the number of parameters `pardim` and number of equations `codim`.

`run_test...` contains the main steps of the evaluation (definition of the parameter space, intrations, plotting) `fval_basic_...` defines the function for the root-finding problem

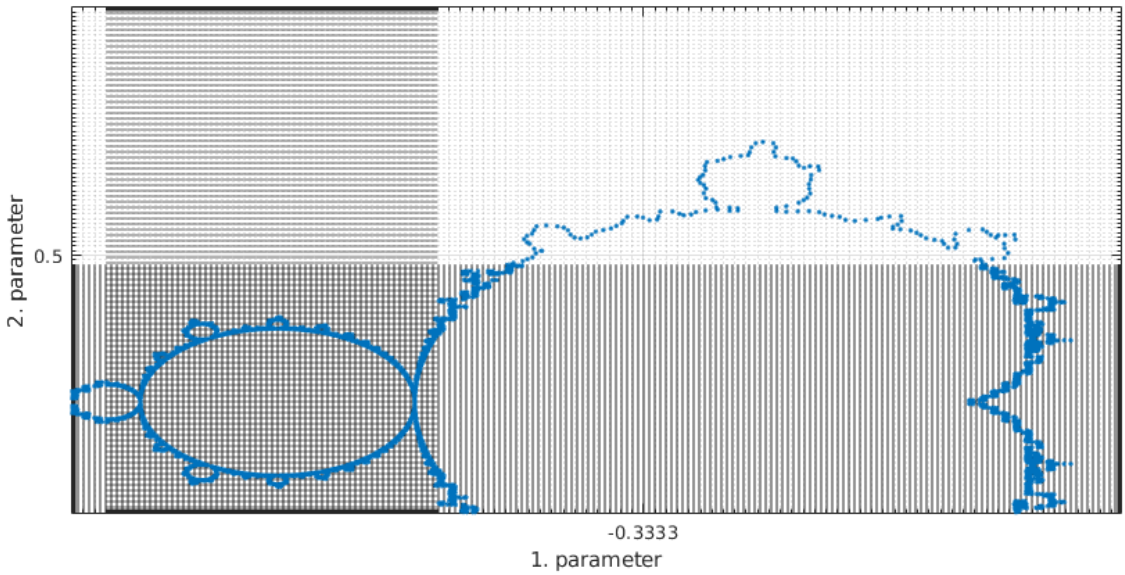
- `run_test_basic_1_pardim1_codim1.m`, `fval_basic_1_pardim1_codim1.m` - solution of `y=sin(x)`
- `run_test_basic_2_pardim2_codim1.m`, `fval_basic_2_pardim2_codim1.m` - the potins fo the function evaluation is presented, and different order (0^{th}, 1^{st}, 2^{nd}) is demonstrated
- `run_test_basic_3_pardim2_codim2.m`, `fval_basic_3_pardim2_codim2.m` - different plotting (triangulated and wireframe) is shown
- `run_test_basic_4_pardim3_codim1.m`, `fval_basic_4_pardim3_codim1.m`
- `run_test_basic_5_pardim3_codim2.m`, `fval_basic_5_pardim3_codim2.m`
- `run_test_basic_6_pardim3_codim3.m`, `fval_basic_6_pardim3_codim3.m`
- `run_test_basic_7_pardim4_codim1.m`, `fval_basic_7_pardim4_codim1.m` - suppressed line connection is presented (which can be very time consuming for higher object dimension)
- `run_test_basic_8_pardim4_codim2.m`, `fval_basic_8_pardim4_codim2.m`
- `run_test_basic_9_pardim4_codim3.m`, `fval_basic_9_pardim4_codim3.m`

Complex examples

Interactive example non-smooth problem (Mandelbrot set)

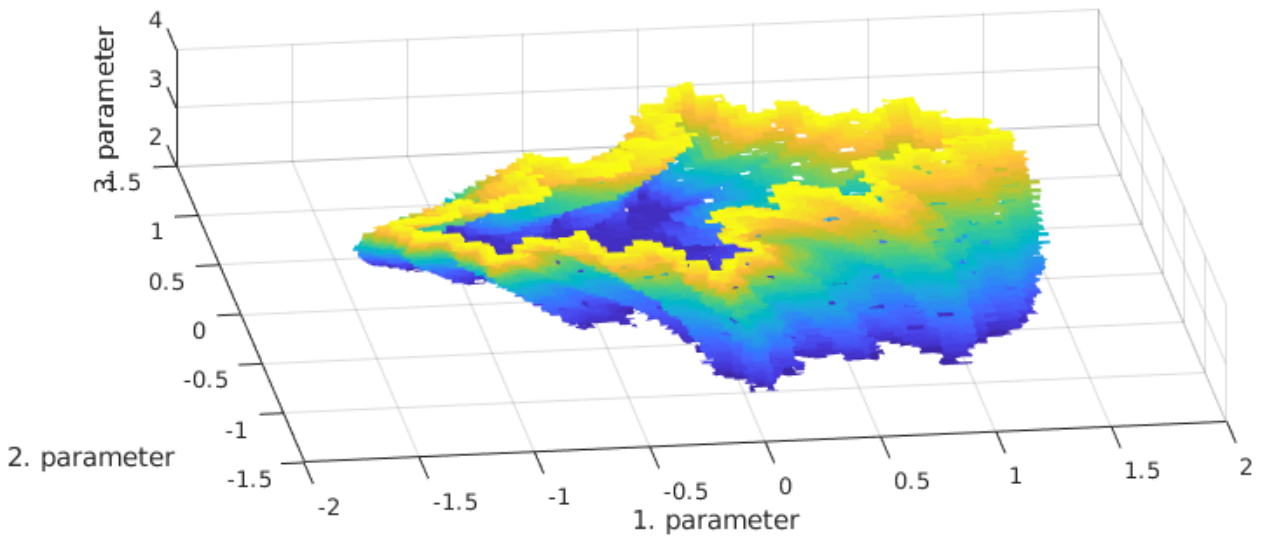
- with grid plot of final resolution
- demonstration of region refinement (only the selected range is refined)
- demonstration of continuation-like behavior (to explore the missing components of the manifold)

`run_test_complex_1_Mandelbrot_set.m`, `fval_complex_1_Mandelbrot_set.m`



Non-smooth problem (3D Julia set)

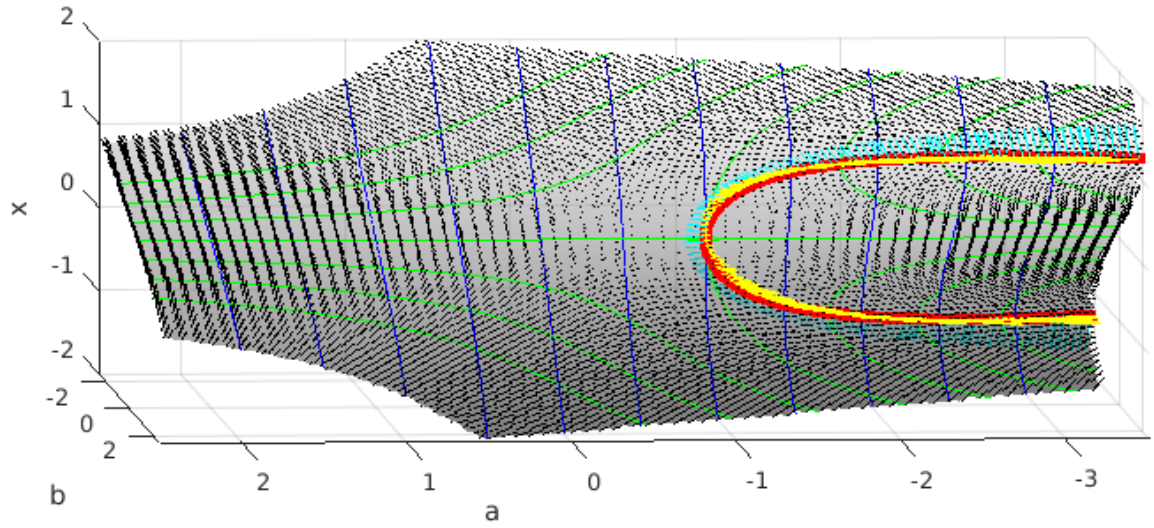
`run_test_complex_1_Julia_set_3d.m`, `fval_complex_1_Julia_set_3d.m`



Equilibrium points of a system with non linearity `a sin(x)+x+b`

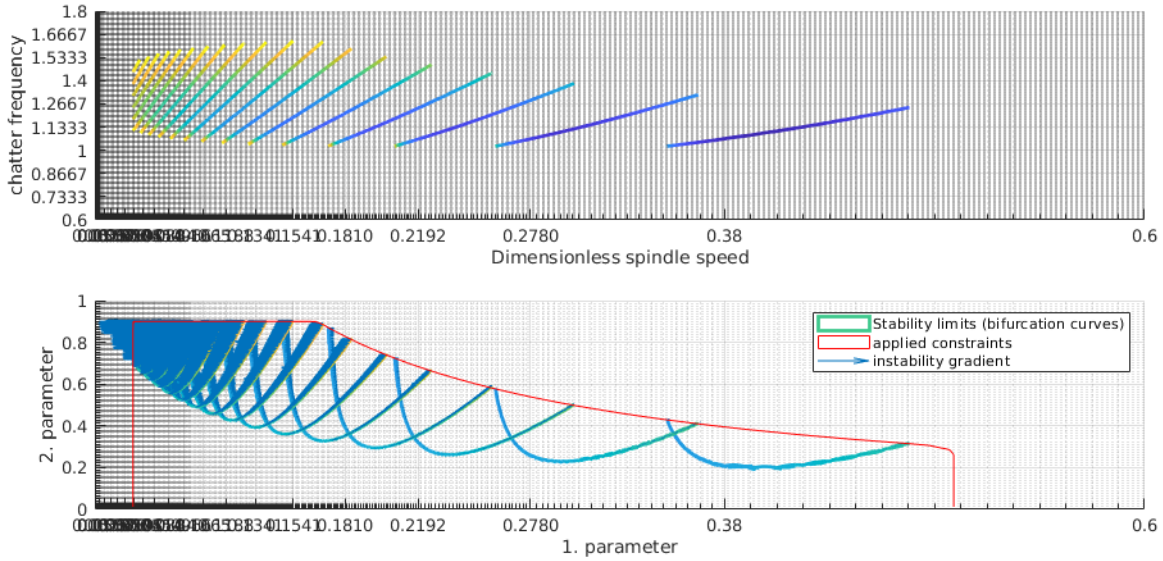
- demonstration of the computed gradients
- different contour lines

`run_test_complex_2_catastrophe_surface.m`, `fval_complex_2_catastrophe_surface.m`



Linear stability chart of the turning process (~delayed oscillator) base on the characteristic equation

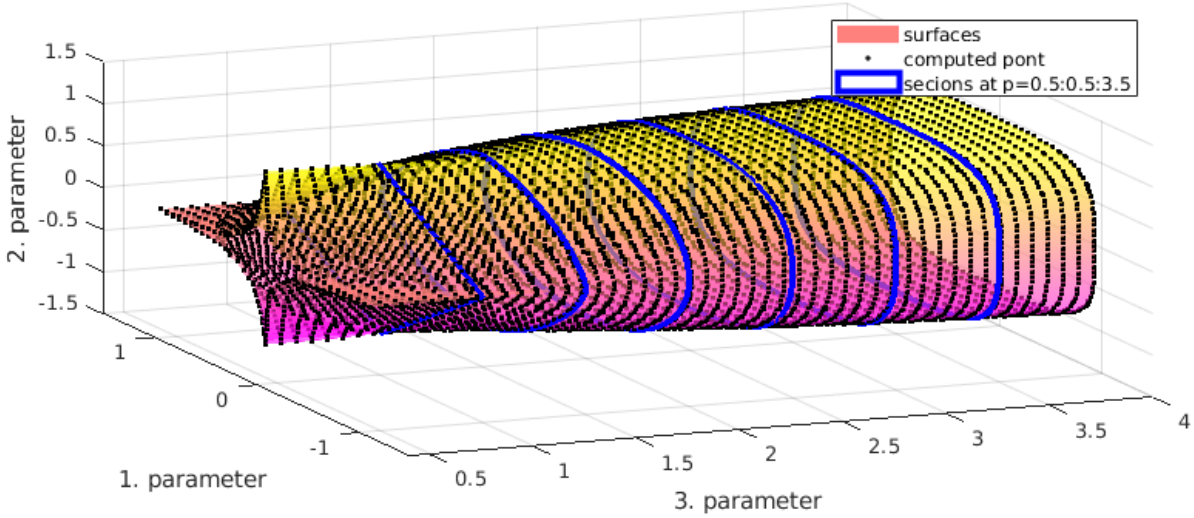
- demonstration of the non-uniform initial grid
- demonstration of the applied extra constraint `run_test_complex_3_turning_stability.m`, `fval_complex_3_turning_stability.m`



Plotting the *Unite-circles* as a function of the power of norm

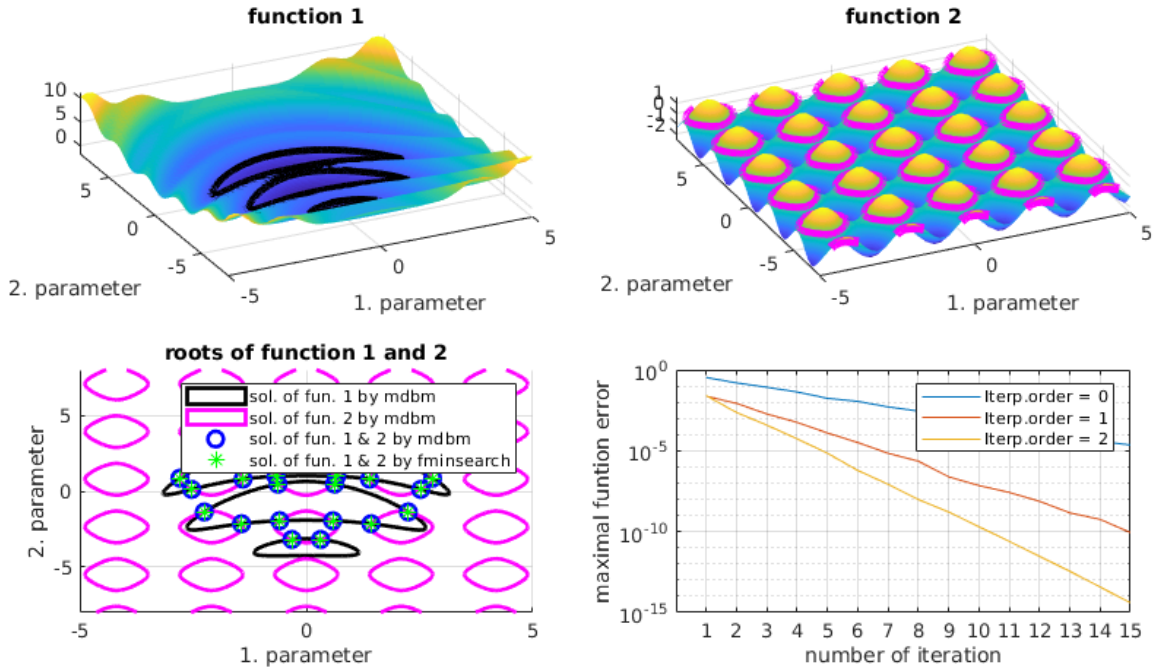
- demonstration of different plotting possibilities

`run_test_complex_4_unit_circles.m`, `fval_complex_4_unit_circles.m`



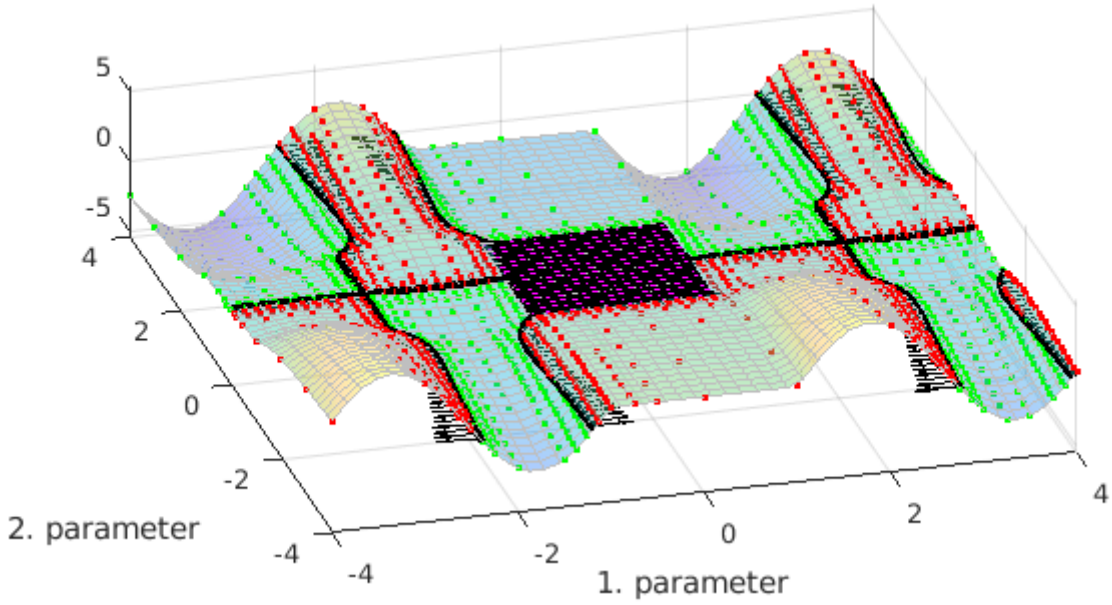
Presentation of the 0^{th}, 1^{st} and 2^{nd} order convergence range of the detected potins>

`run_test_complex_5_transcendent.m`, `fval_complex_5_transcendent.m`



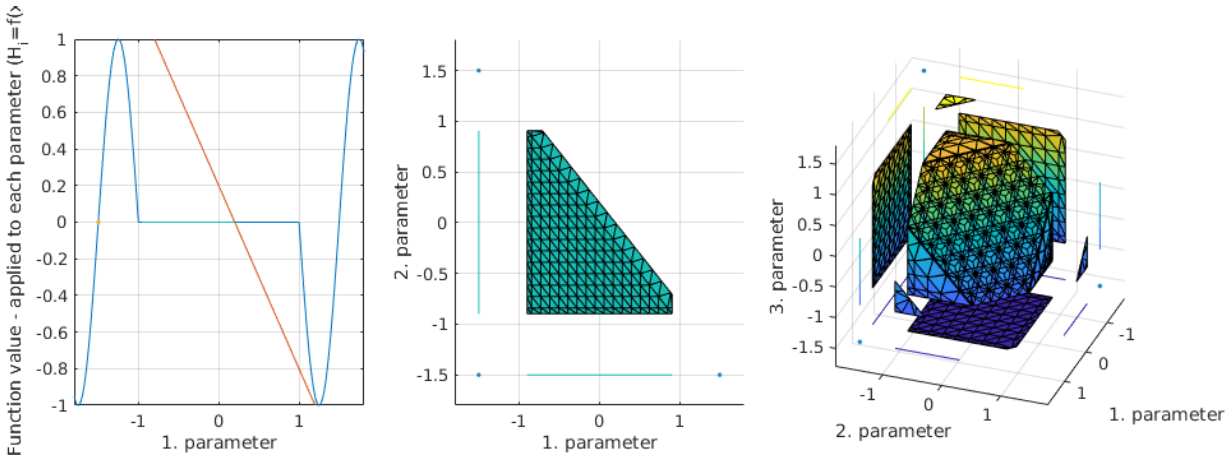
A degenerate example for two parameter, where the topological dimension of one part of the zero manifold is one (curve), while an other part is two (surface)

`run_test_complex_6_degenerate_1.m`, `fval_complex_6_degenerate_1.m`



A degenerate example in 1, 2 and 3 dimension with additional constraint

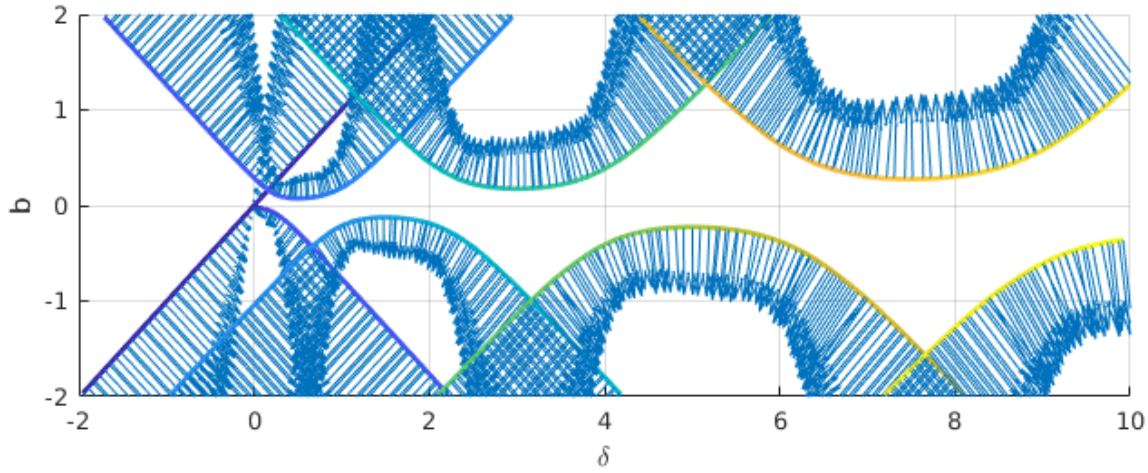
`run_test_complex_7_degenerate_2.m`, `fval_complex_7_degenerate_2.m`



Stability of delayed PD control with instability gradient

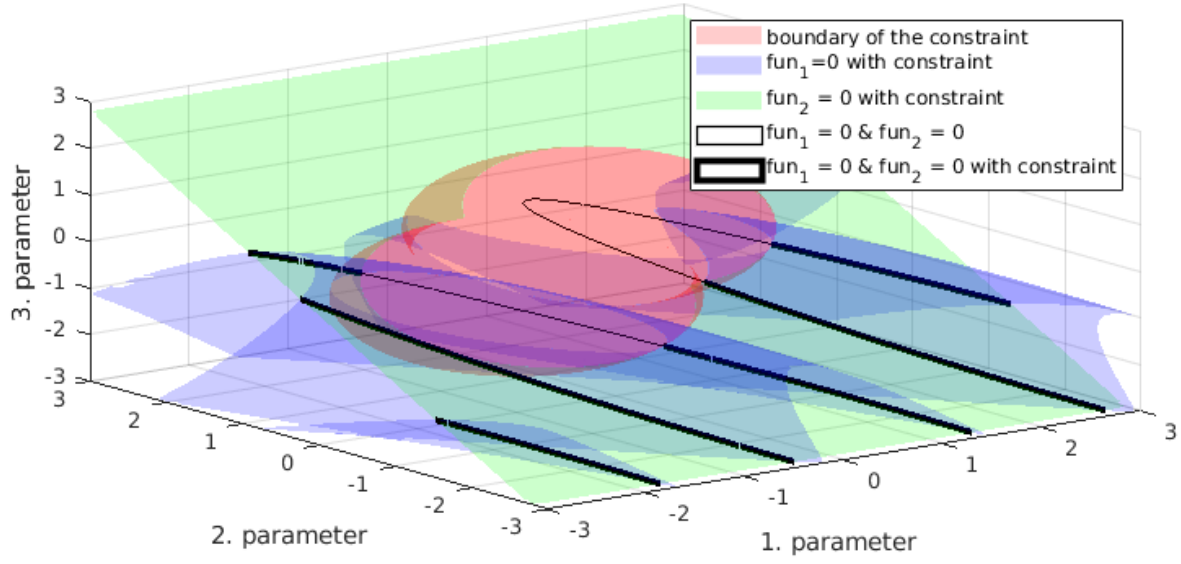
- the gradient of the characteristic equation is used to determine the direction at the stability boundary where more unstable characteristic root are

`run_test_complex_7_delay_oscill_cont.m`, `fval_complex_7_delay_oscill_cont.m`



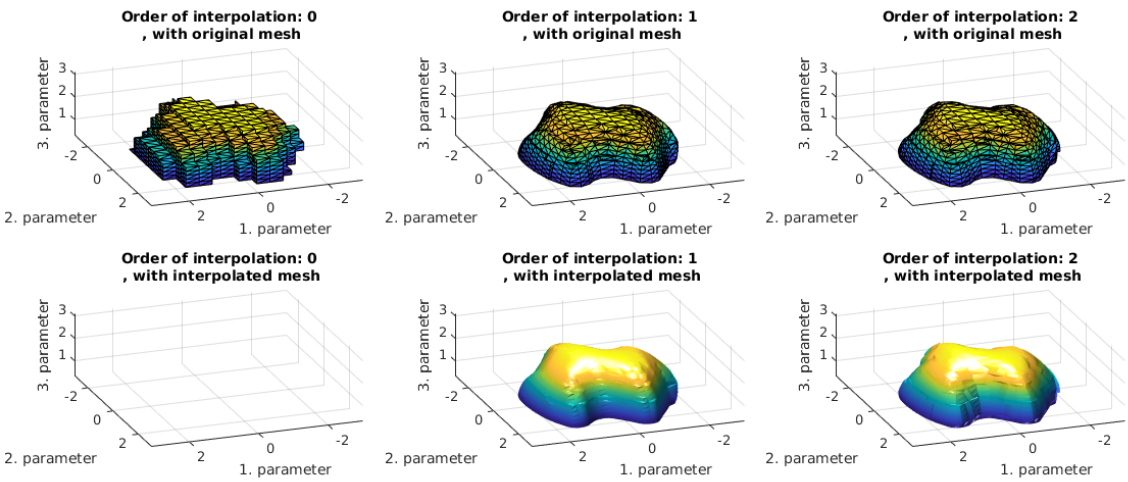
Graphic representation of the intesection of two surfaces with constratint

`run_test_complex_8_constrained.m`, `fval_complex_8_constrained.m`



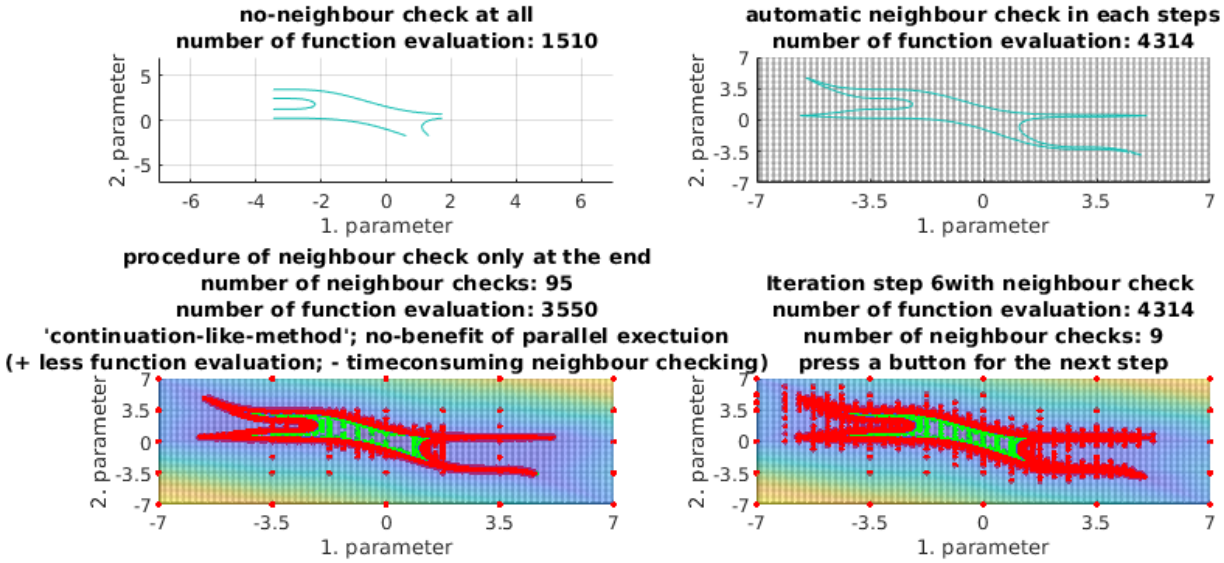
The computed point colud can be interpolater to have a better representation

`run_test_complex_interpplot.m`, `fval_complex_9_interpplot.m`



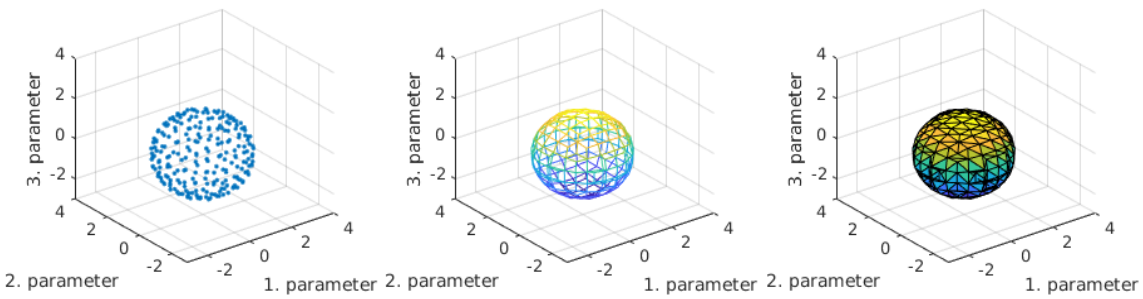
Demonstration of the neighbor checking with an animated interactive example to demonstrate the continuation-like behavior

- `run_test_demonstration_of_neighbour_check.m`



Several different plotting examples

- `run_test_plotting_exampels.m`



MDBM.jl - Julia Package

First add the Multi-Dimensional Bisection Method (MDBM) Package

```
Pkg.add("MDBM")
```

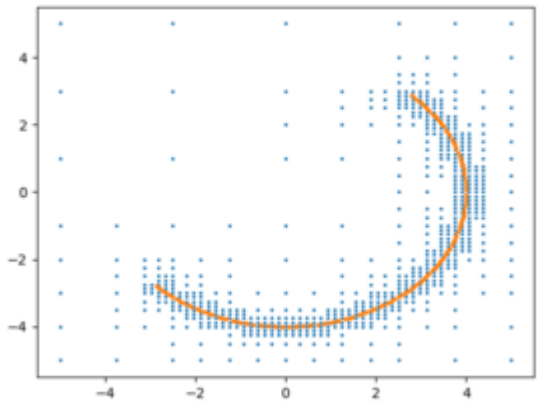
Befor start the example run the following code

```
# using the Multi-Dimensional Bisection Method
using MDBM

#for plotting
using PyPlot;
pygui(true);
```

Example 1

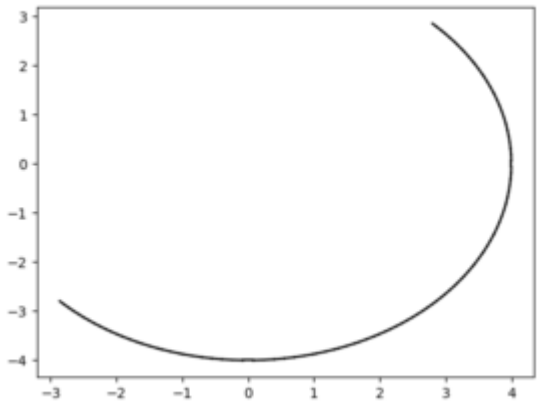
Computation of a circle section defined by the implicit equation `foo(x,y) == 0` and by the constraint `c(x,y) > 0`

```
function foo(x,y)
    x^2.0+y^2.0-2.0^2.0
end
function c(x,y) #only the c>0 domain is analysed
    x-y
end

ax1=Axis([-5,-2.5,0,2.5,5],"x") # initial grid in x direction
ax2=Axis(-5:2:5.0,"y") # initial grid in y direction

mymdbm=MDBM_Problem(foo,[ax1,ax2],constraint=c)
iteration=5 #number of refinements (resolution doubling)
solve!(mymdbm,iteration)


#points where the function foo was evaluated
x_eval,y_eval=getevaluatedpoints(mymdbm)

#interpolated points of the solution (approximately where foo(x,y) == 0
and c(x,y)>0)
x_sol,y_sol=getinterpolatedsolution(mymdbm)

fig = figure(1);clf()
scatter(x_eval,y_eval,s=5)
scatter(x_sol,y_sol,s=5)
```



Perform the line connection

```
myDT1=connect(mymdbm);
for i in 1:length(myDT1)
    dt=myDT1[i]
    P1=getinterpolatedsolution(mymdbm.ncubes[dt[1]],mymdbm)
    P2=getinterpolatedsolution(mymdbm.ncubes[dt[2]],mymdbm)
    plot([P1[1],P2[1]],[P1[2],P2[2]], color="k")
end
```

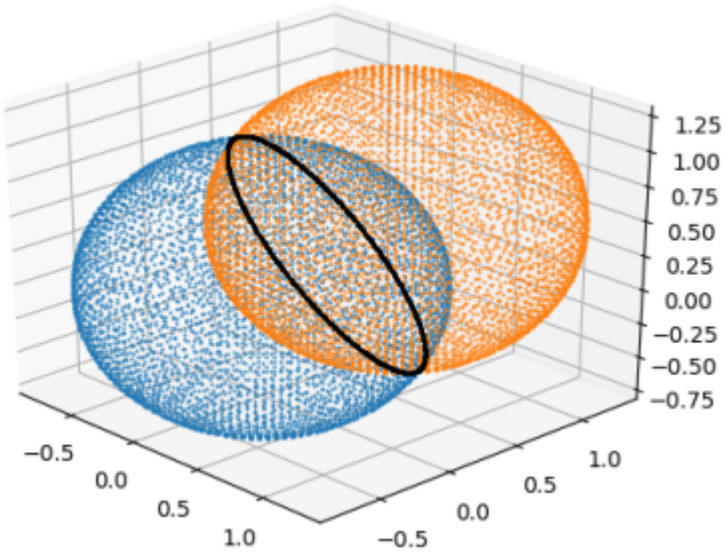


Example 2

Intersection of two sphere in 3D

```
using LinearAlgebra

axes=[-2:2,-2:2,-2:2]

fig = figure(3);clf()


#Sphere1
fS1(x...) = norm(x,2.0)-1
```

```
Sphere1mdbm=MDBM_Problem(fS1,axes)
solve!(Sphere1mdbm,4)
a_sol,b_sol,c_sol=getinterpolatedsolution(Sphere1mdbm)
plot3D(a_sol,b_sol,c_sol,linestyle="", marker=".", markersize=1);


#Sphere2
fS2(x...) = norm(x .- 0.5, 2.0) -1.0

Sphere2mdbm=MDBM_Problem(fS2,axes)
solve!(Sphere2mdbm,4)
a_sol,b_sol,c_sol=getinterpolatedsolution(Sphere2mdbm)
plot3D(a_sol,b_sol,c_sol,linestyle="", marker=".", markersize=1);

#Intersection
fS12(x...) = (fS1(x...), fS2(x...))

Intersectmdbm=MDBM_Problem(fS12,axes)
solve!(Intersectmdbm,6)
a_sol,b_sol,c_sol=getinterpolatedsolution(Intersectmdbm)
plot3D(a_sol,b_sol,c_sol,color="k",linestyle="", marker=".",
markersize=2);
```



Example 3

Mandelbrot set (example for a non-smooth problem)

```
function mandelbrot(x,y)
    c=x+y*im
    z=Complex(zero(c))
    k=0
    maxiteration=1000
    while (k<maxiteration && abs(z)<4)
            z=z^2+c
            k=k+1
```

```
        end
    return abs(z)-2
end

Mandelbrotmdbm=MDBM_Problem(mandelbrot,[-5:2,-2:2])
solve!(Mandelbrotmdbm,9)
a_sol,b_sol=getinterpolatedsolution(Mandelbrotmdbm)
fig = figure(3);clf()
plot(a_sol,b_sol,linestyle="", marker=".", markersize=1)
```